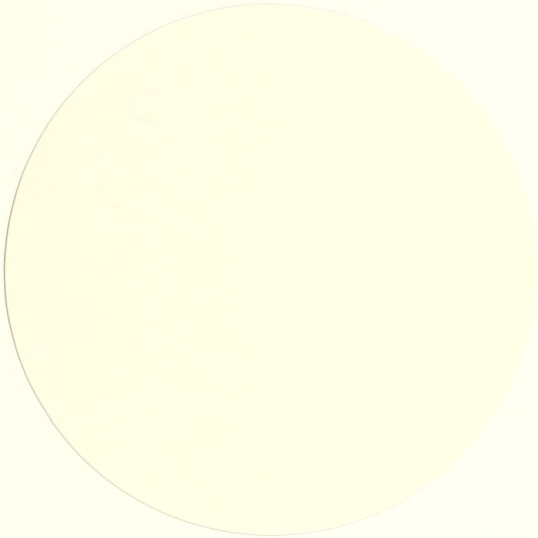
DESIGN OF A GENERAL HIERARCHICAL STORAGE SYSTEM

Stuart E. Madnick

REPORT CISR-6
SLOAN WP-769-75
March 3, 1975

Center for Information Systems Research
Massachusetts Institute of Technology
Alfred P. Sloan School of Management
50 Memorial Drive
Cambridge Massachusetts, 02139



DESIGN OF A GENERAL HIERARCHICAL STORAGE SYSTEM

Stuart E. Madnick

REPORT CISR-6
SLOAN WP-769-75
March 3, 1975

DESIGN OF A GENERAL HIERARCHICAL STORAGE SYSTEM

Stuart E. Madnick
Center for Information Systems Research
Alfred P. Sloan School of Management
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

By extending the concepts used in contemporary two-level hierarchical storage systems, such as cache or paging systems, it is possible to develop an orderly strategy for the design of large-scale automated hierarchical storage systems. In this paper systems encompassing up to six levels of storage technology are studied. Specific techniques, such as page splitting, shadow storage, direct transfer, read through, store behind, and distributed control, are discussed and are shown to provide considerable advantages for increased systems performance and reliability.

DESIGN OF A GENERAL HIERARCHICAL STORAGE SYSTEM

Stuart E. Madnick
Center for Information Systems Research
Alfred P. Sloan School of Management
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

INTRODUCTION

The evolution of computer systems has been marked by a continually increasing demand for faster, larger, and more economical storage facilities. Clearly, if we had the technology to produce ultra-fast limitless-capacity storage devices for miniscule cost, the demand could be easily satisfied. Due to the current unavailability of such a wonderous device, the requirements of high-performance yet low-cost are best satisfied by a mixture of technologies combining expensive high-performance devices with inexpensive lower-performance devices. Such a strategy is often called a hierarchical storage system or multilevel storage system.

CONTEMPORARY HIERARCHICAL STORAGE SYSTEMS

Range of Storage Technologies

Table I indicates the range of performance and cost characteristics for typical current-day storage technologies divided into 6 cost-performance levels. Although this table is only a simplified summary, it does illustrate the fact that there exists a spectrum of devices that span over 6 orders of magnitude (100,000,000%) in both cost and performance.

Locality of Reference

If all references to online storage were equally likely, the use of hierarchical storage systems would be of marginal value at best. For example, consider a hierarchical system, M', consisting of two types of devices, M(1) and M(2), with access times A(1)=.05xA(2) and costs C(1)=20xC(2), respectively. If the total storage capacity S' were equally divided between M(1) and M(2), i.e., S(1)=S(2), then we can determine an overall cost per byte, C', and effective access time, A', as:

$$C' = .5 \times C(1) + .5 \times C(2)$$

$$C' = 10.5 \times C(2) = .5045 \times C(1)$$

and

$$A' = .5 \times A(1) + .5 \times A(2)$$

$$A' = .525 \times A(2) = 10.5 \times A(1)$$

From these figures, we see that the effective system, M', costs about half as much as M(1) but is more than 10 times as slow. On the other hand, M' is almost twice as fast as M(2), but it costs more than 10 times as much per byte.

Fortunately, most actual programs and applications cluster their references so that, during any interval of time, only a small subset of the information is actually used. This phenomenon is known as locality of reference. If we could keep the "current" information in a small amount of M(1), we could produce hierarchical storage systems with an effective access time close to M(1) but an overall cost per byte close to M(2).

The actual performance attainable by such hierarchical storage systems is addressed by the other papers in this session[12,14,20,23] and the general literature[2,3,4,6,7,8,10,11,13,15,18,19,21,22]. In many actual systems it has been possible to find over 90% of all references in M(1) even though M(1) was much smaller than M(2)[13,15].

Automated Hierarchical Storage Systems

There are at least three ways in which the locality of reference can be exploited in a hierarchical storage system: static, manual, or automatic.

If the pattern of reference is constant and predictable over a long period of time (e.g., days, weeks, or months), the information can be statically allocated to specific storage devices so that the most frequently referenced information is assigned to the highest performance devices.

Storage Level	Random Access Time	Transfer Rate (btyes/second)	Cost per Byte	Technology
1. Cache	50 ns	100M	100¢	Semiconductor RAM
2. Main	1 µs	16M	10¢	Semiconductor RAM, Ferrite core
3. Block	50 µs	8M	2¢	Semiconductor shift registers, Bulk ferrite core, Charge-coupled devices, Magnetic bubbles
4. Backing	1 ms	2M	.5¢	Fixed-head disks and drums, Charge-coupled devices, Magnetic bubbles
5. Secondary	50 ms	1M	.01¢	Moving-head disks
6. Mass	1 sec	1M	.005¢	Automated tape-handlers, Laser devices

Table I.
Spectrum of Storage Device Technologies

If the pattern of reference is not constant but is predictable, the programmer can manually (i.e., by explicit instructions in the program) cause frequently referenced information to be moved onto high performance devices while being used. Such manual control places a heavy burden on the programmer to understand the detailed behavior of the application and to determine the optimum way to utilize the storage hierarchy. This can be very difficult if the application is complex (e.g., multiple access data base system) or is operating in a multiprogramming environment with other applications.

In an automated storage hierarchy, all aspects of the physical information organization and distribution and movement of information are removed from the programmer's responsibility. The algorithms that manage the storage hierarchy may be implemented in hardware, firmware, system software or some combination.

Contemporary Automated Hierarchical Storage Systems

To date, most implementations of automated hierarchical storage systems have been large-scale cache systems or paging systems. Typically, cache systems, using a combination of level 1 and level 2 storage devices managed by specialized hardware, have been used in large-scale high-performance computer systems[4,10,23]. Paging systems, also called virtual storage systems, usually employ level 2 and level 4 storage devices managed mostly by system software with limited support from specialized hardware and/or firmware[5,11,12].

Impact of New Technology

As a result of new storage technologies, especially for levels 3 and 6, as well as substantially reduced costs for implementing the necessary hardware control functions, many new types of hierarchical storage systems have evolved. Several of these advances are discussed in this session.

Bill Strecker[23] explains how the cache system concept can be extended for use in high-performance minicomputer systems.

Bernard Greenberg and Steven Weber [12] discuss the design and performance of a three-level paging system used in the Multics Operating System (levels 2, 4, and 5 in the earlier Honeywell 645 implementation; levels 2, 3, and 5 in the more recent Honeywell 68/80 implementation).

Suran Ohnigian[20] analyzes the handling of data base applications in a storage hierarchy.

Clay Johnson[14] explains how a new level 6 technology is being used in conjunction with level 5 devices to form a very large capacity hierarchical storage system, the IBM 3850 Mass Storage System.

The various automated hierarchical storage systems discussed in this session are depicted in Figure 1.

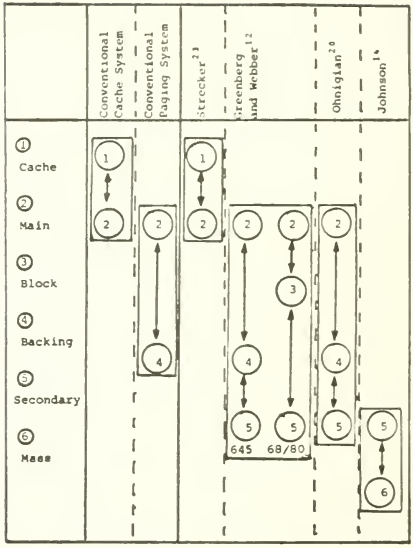


Figure 1

Various Automated Hierarchical Storage Systems

GENERAL HIERARCHICAL STORAGE SYSTEM

All of the automated hierarchical storage systems described thus far have only dealt with two or three levels of storage devices (the Honeywell 68/80 Multics system actually encompasses four levels since it contains a cache memory which is partially controlled by the system software). Furthermore, many of these systems were initially conceived and designed several years ago without the benefit of our current knowledge about memory and processor capabilities and costs. In this section the structure of a general automated hierarchical storage system is outlined (see Figure 2).

Continuous and Complete Hierarchy

An automated storage hierarchy must consist of a continuous and complete hierarchy that encompasses the full range of storage levels. Otherwise, the user would be forced to rely on manual or semi-automatic storage management techniques to deal with the storage levels that are not automatically managed.

Page Splitting and Shadow Storage

The average time, Tm, required to move a page, a unit of information, between two levels of the hierarchy consists of the sum of (1) the average access time, T, and (2) the transfer time, which is the product of the transfer rate, R, and the size of a page, N. Thus, $Tm = T + R \times N$.

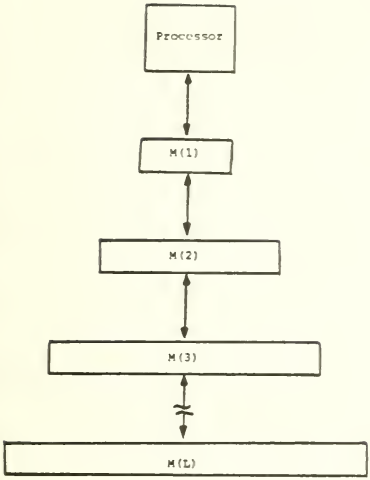


Figure 2

General Automated
Hierarchical Storage System

Choice of Page Size. By examining the representative devices indicated in Table I, we see that access times vary much more than transfer rates. As a result, Tm is very sensitive to the transfer unit size, N, for devices with small values of T (e.g., levels 1 and 2). Thus, for these devices small page sizes such as N=16 or 32 bytes are typically used. Conversely, for devices with large values for T, the marginal cost of increasing the page size is quite small, especially in view of the benefits possible due to locality of reference. Thus, page sizes such as N=4096 bytes are typical for levels 4 and 5. Much larger units of information transfer are used for level 6 devices.

Since the marginal increase in Tm decreases monotonically as a function of storage level, the number of bytes transferred between levels should increase correspondingly. In order to simplify the implementation of the system and to be consistent with the normal mappings from processor address references to page addresses, it is desirable that all page sizes be a power of two. The choice for each page size depends upon the characteristics of the programs to be run and the effectiveness of the overall storage system. Preliminary measurements indicate that a ratio of 4:1 or 8:1 between levels is reasonable. Meade[19] has reported similar findings. Figure 3 indicates possible transfer unit sizes between the 6 levels presented in Table I. The particular choice of size will, of course, depend upon the particulars of the devices.

Level	Transfer Unit
0. Processor	
	4
1. Cache	
	32
2. Main	
	256
3. Block	
	2048
4. Backing	
	16384
5. Secondary	
	131072
6. Mass	

Figure 3

Sample Transfer Unit Sizes

Page Splitting. Now let us consider the actual movement of information in the storage hierarchy. At time t, the processor generates a reference for logical address a. Assume that the corresponding information is not currently stored in M(1) or M(2) but is found in M(3). For simplicity, assume that page sizes are doubled as we go down the hierarchy (i.e., N(2,3)=2xN(1,2), N(3,4)=2xN(2,3)=4xN(1,2),etc.; see Figure 4). The page of size N(2,3) containing a is copied from M(3) to M(2). M(2) now contains the needed information, so we repeat the process. The page of size N(1,2) containing a is copied from M(2) to M(1). Now, finally, the page of size N(0,1) containing a is copied from M(1) and forwarded to the processor. In the process the page of information is split repeatedly as it moves up the hierarchy.

Shadow Storage. As a result of the splitting process, the page of size N(0,1) that is received by the processor has left a shadow consisting of itself and its adjacent pages behind in all the lower levels. Presumably, if the program or application exhibits locality of reference, many of these shadow pages will be referenced shortly afterward and be moved further up in the hierarchy also.

Copying of Pages. In the strategy presented, pages are actually copied as they move up the hierarchy; a page at level i has one copy of itself in each of the levels j > i. Since processor "read" requests substantially outnumber "write" requests for most applications (e.g., by more than 5:1 in some measured programs), the contents of pages are seldom changed. Thus, if a page has not been changed and is selected to be removed from one level to a lower level, it need not be actually transferred since a valid copy already exists in the lower level. The contents of any level of the hierarchy is always a subset of the information contained in the next lower level.

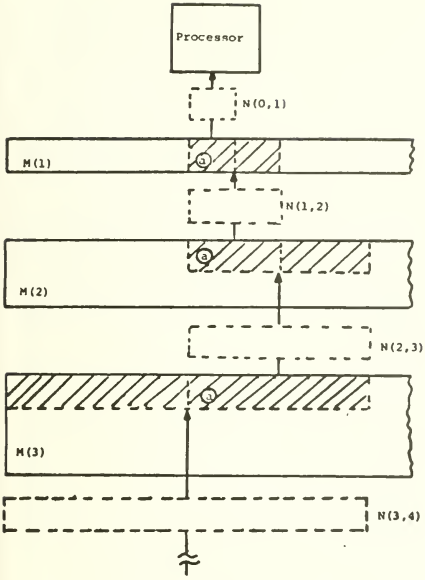


Figure 4

Page Splitting and
Shadow Storage

Direct Transfer

In the descriptions above it is implied that information is directly transferred between adjacent levels. By comparison, most present multiple level storage systems are based upon an indirect transfer (e.g., the Multics Page Multilevel system[12]). In an indirect transfer system, all information is routed through level 1 or level 2. For example, to move a page from level n-1 to level n, the page is moved from level n-1 to level 1 and then from level 1 to level n.

Clearly, an indirect approach is undesirable since it requires extra page movement and its associated overhead as well as consuming a portion of the limited M(1) capacity in the process. In the past such indirect schemes were necessary since all interconnections were radial from the main memory or processor to the storage devices. Thus, all transfers had to be routed through the central element. Furthermore, due to the differences in transfer rate between devices, especially electro-mechanical devices that must oprate at a fixed transfer rate, direct transfer was not possible.

Based upon current technology, these problems can be solved. Many of the newer storage devices are non-electromechanical (i.e., strictly electrical). In these cases the transfer rates can be synchronized to allow direct transfer. For electromechanical devices, direct transfer is possible if transfer rates are similar or a small scratchpad buffer, such as low-cost semiconductor shift registers, are used to enable synchronization between the devices. Direct transfer is provided between the level 6 and level 5 storage devices in the IBM 3850 Mass Storage System[14].

Read Through

A possible implementation of the page splitting strategy could be accomplished by transferring information from level i to the processor (level 0) in a series of sequential steps: (1) transfer page of size N(i-1,i) from level i to level i-1, and then (2) extract the appropriate page subset of size N(i-2, i-1) and transfer it from level i-1 to level i-2, etc. Under such a scheme, a transfer from level i to the processor would consist of a series of i steps. This would result in a considerable delay, especially for certain devices that require a large delay for reading recently written information (e.g., rotational delay to reposition electromechanical devices).

The inefficiencies of the sequential transfers can be avoided by having the information read through to all the upper levels simultaneously. For example, assume the processor references logical address a which is currently stored at level i (and all lower levels, of course). The controller for M(i) outputs the N(i-1,i) bytes that contain a onto the data buses along with their corresponding addresses. The controller for M(i-1) will accept all of these N(i-1,i) bytes and store them in M(i-1). At the same time, the controller for M(i-2) will extract the particular N(i-2, i-1) bytes that it wants and store them in M(i-2). In a like manner, all of the controllers can simultaneously extract the portion that is needed. Using this mechanism, the processor (level 0) extracts the N(0,1) bytes that it needs without any further delays -- thus the information from level i is read through to the processor without any delays. This process is illustrated in Figure 5.

The read through mechanism also offers some important reliability, availability, and serviceability advantages. Since all storage levels communicate anonymously over the data buses, if a storage level must be removed from the system, there are no changes needed. In this case, the information is "read through" this level as if it didn't exist. No changes are needed to any of the other storage levels or the storage management algorithms although we would expect the performance to decrease as a result of the missing storage level. This reliability strategy is employed in most current-day cache memory systems[10].

Store Behind

Under steady-state operation, we would expect all the levels of the storage hierarchy to be full (with the exception of the lowest level, L). Thus, whenever a page is to be moved into a level, it is necessary to remove a current page. If the page selected for removal has not been changed by means of a processor write, the new page can be immediately stored into the level since a copy of the page to be removed already exists in the next lower level of the hierarchy. But, if the processor performs a write operation, all levels that contain a copy of the information being modified must be updated. This can be accomplished in either of at least three basic ways: (1) store through, (2) store replacement, or (3) store behind.

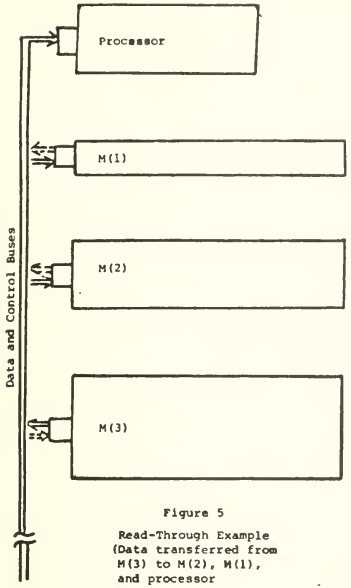


Figure 5

Read-Through Example
(Data transferred from M(3) to M(2), M(1), and processor simultaneously)

Store Through. Under a store through strategy, all levels are updated simultaneously. This is the logical inverse of the read through strategy but it has a crucial distinction. The store through is limited by the speed of the lowest level, L, of the hierarchy, whereas the read through is limited by the speed of the highest level containing the desired information. Store through is efficient only if the access time of level L is comparable to the access time of level 1, such as in a two-level cache system (e.g., it is used in the IBM System/370 Models 158 and 168).

Store Replacement. Under a store replacement strategy, the processor only updates M(1) and marks the page as "changed." If such a changed page is later selected for removal from M(1), it is then moved to the next lower level, M(2), immediately prior to being replaced. This process occurs at every level and, eventually, level L will be updated but only after the page has been selected for removal from all the higher levels. Due to the extra delays caused by updating changed pages before replacement, the effective access time for reads is increased. Various versions of store replacement are used in most two-level paging systems since it offers substantially better performance than store through for slow storage devices (e.g., disks and drums).

Store Behind. In both strategies above, the storage system was required to perform the update operation at some specific time, either at the time of write or replacement. Once the information has been stored into M(1), the processor doesn't really care how or when the other copies are updated. Store behind takes advantage of this degree of freedom. The maximum transfer capability between levels is rarely maintained since, at any instant of time, a storage level may not have any outstanding requests for service or it may be waiting for proper positioning to service a pending request. During these "idle" periods, data can be transferred down to the next lower level of the storage hierarchy without adding any significant delays to the read or store operations. Since these "idle" periods are usually quite frequent under normal operation, there can be a continual flow of changed information down through the storage hierarchy.

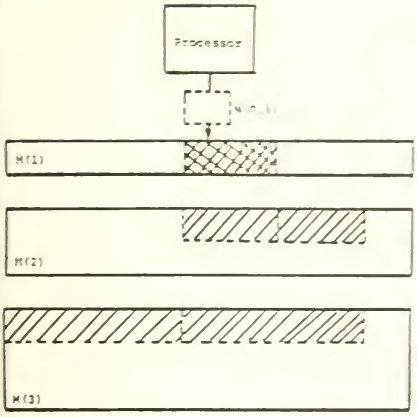
A variation of this store behind strategy is used in the Multics Page Multilevel system[12] whereby a certain number of pages at each level are kept available for immediate replacement. If the number of replaceable pages drops below the threshold, changed pages are selected to be updated to the next lower level. The actual writing of these pages to the lower level is scheduled to take place when there is no read request to be done.

The store behind strategy can also be used to provide high reliability in the storage system. Ordinarily, a changed page is not purged from a level until the next lower level acknowledges that the corresponding page has been updated. We can extend this approach to require two levels of acknowledgement. For example, a changed page is not removed from M(1) until the corresponding pages in both M(2) and M(3) have been updated. In this way, there will be at least two copies of each changed piece of information at levels M(i) and M(i+1) in the hierarchy. If any level malfunctions or must be serviced, it can be removed from the hierarchy without causing any information to be lost. There are two exceptions to this process, levels M(1) and M(L). To completely safeguard the reliability of the system, it may be necessary to store duplicate copies of information at these levels. Figure 6 illustrates this process.

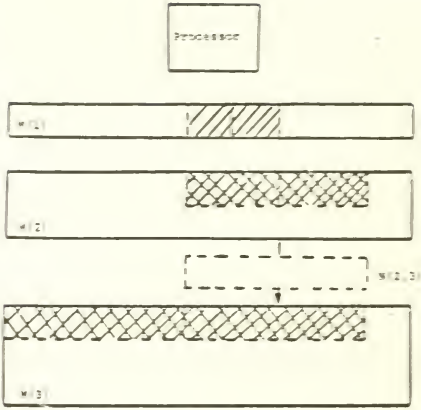
Distributed Control

There is a substantial amount of work required to manage the storage hierarchy. It is desirable to remove as much as possible of the storage management from the concern of the processor. In the hierarchical storage system described in this paper, all of the management algorithms can be operated based upon information that is local to a level or, at most, in conjunction with information from neighboring levels. Thus, it is possible to distribute control of the hierarchy into the levels, this also facilitates parallel and asychronous operation in the hierarchy (e.g., the store behind algorithm).

The independent control facilities for each level can be accomplished by extending the functionality of conventional device controllers. Most current-day sophisticated device controllers are actually microprogrammed processors[1] capable of performing the storage management function. The IBM 3850 Mass Storage System uses such a controller[14].



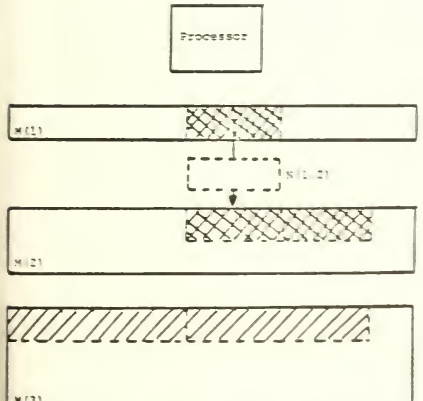
(a) Processor stores into M(1), corresponding page marked "changed."



(c) At a later time, the corresponding page in M(3) is updated and marked "changed." Since copies of the changed information exists in both M(2) and M(3), the "changed" indicator can be reset in M(2) and that page may be replaced if necessary.

Figure 4

Example of the Store-Behind Process



(b) At a later time, the corresponding page in M(2) is updated and marked "changed."

CONCLUDING COMMENTS

In this paper various forms of contempory hierarchical storage system are described. The other papers[1,2,14,20,23] in this session are shown to address various levels of the storage hierarchy. Finally by extrapolating from current hierarchical storage systems and the trends in memory and processor technology, it is shown that a general scheme for automated management of a complete hierarchical storage system can be developed. It is the intent of the general design presented in this paper to stimulate further research and development of future systems (e.g., reference 16).

REFERENCES

1. Ahearn, G. R., Y. Dishon, and R. N. Snively, "Design Innovations of the IBM 3830 and 2835 Storage Control Units," IBM Journal of Research and Development, Vol. 16, No. 1, pp. 11-18, January, 1972.

2. Anacker, W. and C. P. Wong, "Performance Evaluation of Computer Systems with Memory Hierarchies," IEEE-TC, Vol. EC-16, No. 6, pp. 765-773, December 1967.

3. Arora, S. R. and A. Gallo, "Optimal Sizing, Loading and Re-loading in a Multi-level Memory Hierarchy System," SJCC, Vol. 38, pp. 337-344, 1971.

4. Bell, Gorden C. and David Casasent, "Implementation of a Buffer Memory in Minicomputers," Computer Design, pp. 83-89, November 1971.

5. Beneoussan, A., C. T. Clingen, and R. C. Daley, "The Multics Virtual Memory," Second ACM Symposium on Operating Systems Principles, Princeton University, pp. 30-42, October 1969.

6. Brawn, B. S., and F. G. Gustavson, "Program Behavior in a Paging Environment," FJCC, Vol. 33, Pt. 11, pp. 1019-1032, 1968.

7. Chu, W. W., and H. Opderbeck, "Performance of Replacement Algorithms with Different Page Sizes," Computer, Vol. 7, No. 11, pp. 14-21, November 1974.

8. Coffman, E.B., Jr., and L. C. Varian, "Further Experimental Data on the Behavior of Programs in a Paging Environment," CACM, Vol. 11, No. 7, pp. 471-474, July 1968.

9. Considine, J. P., and A. M. Weis, "Establishment and Maintenance of a Storage Hierarchy for an On-line Data Base under TSS/360," FJCC, Vol. 35, pp. 433-440, 1969.

10. Conti, C. J., "Concepts for Buffer Storage," IEEE Computer Group News, pp. 6-13, March 1969.

11. Denning, P. J., "Virtual Memory," ACM Computing Surveys, Vol. 2, No. 3, pp. 153-190, September 1970.

12. Greenberg, Bernard S., and Steven H. Webber, "Multics Multilevel Paging Hierarchy," IEEE INTERCON, 1975.

13. Hatfield, D. J., "Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance," IBM Journal of Research and Development, Vol. 16, No. 1, pp. 58-66, January 1972.

14. Johnson, Clay, "IBM 3850 -- Mass Storage System," IEEE INTERCON, 1975.

15. Madnick, S. E., "Storage Hierarchy Systems," MIT Project MAC Report TR-105, Cambridge, Massachusetts, 1973.

16. Madnick, S. E., "INFOPLEX -- Hierarchical Decomposition of a Large Information Management System Using a Microprocessor Complex," NCC, May 1975.

17. Madnick, S. E., and J. J. Donovan, Operating Systems, McGraw-Hill, New York, 1974.

18. Mattson, R., et al., "Evaluation Techniques for Storage Hierarchies," IBM Systems Journal, Vol. 9, No. 2, pp. 78-117, 1970.

19. Meade, R. M., "On Memory System Design," FJCC, Vol. 37, pp. 33-44, 1970.

20. Ohnigian, Suran, "Random File Processing in a Storage Hierarchy," IEEE INTERCON, 1975.

21. Ramamoorthy, C. V. and K. M. Chandy, "Optimization of Memory Hierarchies in Multiprogrammed Systems," JACM, Vol. 17, No. 3, pp. 426-445, July 1970.

22. Spirn, J. R., and P. J. Denning, "Experiments with Program Locality," FJCC, Vol. 41, Pt. 1, pp. 611-621, 1972.

23. Strecker, William, "Cache Memories for Minicomputer Systems," IEEE INTERCON, 1975.

24. Wilkes, M. V., "Slave Memories and Dynamic Storage Allocation," IEEE-TC, Vol. EC-14, No. 2, pp. 270-271, April 1965.

25. Withington, F. G., "Beyond 1984: A Technology Forecast," Datamation, Vol. 21, No. 1, pp. 54-73, January 1975.